this 变量

目录

- 顶层的 this
- 函数内部的 this

Node 应用的顶层变量是 `global`，对应浏览器的 `window` 变量。

顶层的 this

在 REPL 环境，顶层的 `this` 就指向 `global`。

```
> global === this
true
```

顶层变量是 `global` 和 `this` 的属性。

```
> var foo = "bar";
> this.foo
bar
> global.foo
bar
```

上面代码中，`foo` 是一个顶层变量，自动生成了 `this.foo` 和 `global.foo` 两个属性。

在模块环境，顶层的 `this` 指向当前模块，即 `module.exports`，默认是一个空对象，与 `global` 不是同一个对象。

```
// 模块环境
console.log(this) // {}
console.log(this === global) // false
```

模块内部的顶层变量，不会自动成为 `global` 和 `this` 的属性。

```
// 模块环境
var foo = "bar";
console.log(this.foo); // undefined
console.log(global.foo); // undefined
```

上面代码中，顶层变量 `foo` 并不会生成 `this.foo` 和 `global.foo` 两个属性。这是因为 `foo` 是模块内部的变量，不是全局有效，因此不是 `global` 的属性，而 `this` 是当前的模块对象，`this.foo` 代表模块实例的属性，这跟变量 `foo` 是两回事情。

另外，如果声明变量的时候，不使用 `var` 命令，而是直接赋值，那么该变量在 REPL 环境下将成为 `global` 和 `this` 的属性，在模块环境将只成为 global 的属性。

```
// REPL 环境
> foo = "bar";
> global.foo
bar
> this.foo
bar

// 模块环境
foo = "bar";
console.log(this.foo); // undefined
console.log(global.foo); // bar
```

函数内部的 this

直接执行一个函数（不使用 `new` 命令），函数内部的 `this` 指向 `global`， REPL 环境和模块环境都是如此。

```
foo = "bar";

function testThis () {
  this.foo = "foo";
}

console.log(global.foo); // bar
testThis();
console.log(global.foo); // foo
```

如果是严格模式，函数内部的 `this` 返回 `undefined` 。

```
foo = "bar";

function testThis() {
  "use strict";
  this.foo = "foo";
}

console.log(this.foo); // "bar"
testThis();  // TypeError: Cannot set property 'foo' of undefined
```

如果使用 `new` 命令调用某个函数，该函数就变成了构造函数，函数内部的 `this` 指向新建的实例对象。

版本号 #2
由 壳壳虫 创建于 2017-07-04 13:37:26 UTC
由 壳壳虫 更新于 2017-07-04 13:37:26 UTC